Course: Deep Learning

Name: SAYEDPEDRAM HAERI BOROUJENI

Instructor: Dr. Feng Luo

Home Work: Number One

Part 2: Optimization

1. Import My Packages

In [26]:
```
import os
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.backends.cudnn as cudnn
import torchvision.transforms as transformtransforms
import torchvision.transforms.functional as TF
from torchvision import models
from torchsummary import summary
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import ToPILImage
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
from tqdm import tqdm
import cv2
import copy
import math
import random
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from glob import glob

os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
TORCH_CUDA_ARCH_LIST="8.6"

Project_PATH = os.path.dirname(os.path.abspath('__file__'))
outputs_dir = Project_PATH + '/Desktop/Deep Learning HW1/'
model_path = Project_PATH + '/save_models/'
```

2. My Device

In [13]:
```
device_default = torch.cuda.current_device()
torch.cuda.device(device_default)
device = torch.device("cuda")
```

```
print("torch.cuda.is_available:", torch.cuda.is_available())
print("torch.cuda.device_count:", torch.cuda.device_count())
print("torch.cuda.current_device:", torch.cuda.current_device())
print("torch.cuda.get_device_name:", torch.cuda.get_device_name(device_default))
print("torch.version.cuda:", torch.version.cuda)
print("torch.version:", torch.__version__)
print("torch.cuda.arch_list:", torch.cuda.get_arch_list())
```

```
torch.cuda.is_available: True
torch.cuda.device_count: 1
torch.cuda.current_device: 0
torch.cuda.get_device_name: NVIDIA RTX A5000
torch.version.cuda: 11.3
torch.version: 1.11.0
torch.cuda.arch_list: ['sm_37', 'sm_50', 'sm_60', 'sm_61', 'sm_70', 'sm_75', 'sm_80',
'sm_86', 'compute_37']
```

3. Optimization Process

In [14]:

```
# DNN model for MNist Dataset with 3 layers
class DNN_MNIST(nn.Module):
    def __init__(self):
        super(DNN_MNIST, self).__init__()
        self.layer1 = nn.Sequential(nn.Linear(28*28, 32),nn.ReLU(True))
        self.layer2 = nn.Sequential(nn.Linear(32, 16),nn.ReLU(True))
        self.layer3 = nn.Sequential(nn.Linear(16, 10))
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        return x

device = torch.device("cuda")
Model_DNN_MNIST = DNN_MNIST().to(device)
summary(Model_DNN_MNIST, input_size=(1,28*28))

# CNN model for CIFAR Dataset with 6 layers
class CNN_CIFAR(nn.Module):
    def __init__(self):
        super(CNN_CIFAR, self).__init__()
        self.layer1 = nn.Sequential(nn.Conv2d(3, 10, 3),nn.BatchNorm2d(10),nn.ReLU(Tru
        self.layer2 = nn.Sequential(nn.Conv2d(10, 16, 3),nn.BatchNorm2d(16),nn.ReLU(Tr
        self.layer3 = nn.Sequential(nn.Conv2d(16, 32, 3),nn.BatchNorm2d(32),nn.ReLU(Tr
        self.layer4 = nn.Sequential(nn.Linear(32*4*4, 64),nn.BatchNorm1d(64),nn.ReLU(T
        self.layer5 = nn.Sequential(nn.Linear(64, 16),nn.BatchNorm1d(16),nn.ReLU(True)
        self.layer6 = nn.Sequential(nn.Linear(16, 10))
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = x.view(x.size()[0], -1)
        x = self.layer4(x)
        x = self.layer5(x)
        x = self.layer6(x)
        return x

device = torch.device("cuda")
Model = CNN_CIFAR().to(device)
summary(Model, input_size=(3,32,32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 32]          25,120
              ReLU-2                [-1, 1, 32]               0
            Linear-3                [-1, 1, 16]             528
              ReLU-4                [-1, 1, 16]               0
            Linear-5                [-1, 1, 10]             170
================================================================
Total params: 25,818
Trainable params: 25,818
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.10
Estimated Total Size (MB): 0.10
----------------------------------------------------------------
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 10, 30, 30]             280
       BatchNorm2d-2           [-1, 10, 30, 30]              20
              ReLU-3           [-1, 10, 30, 30]               0
         MaxPool2d-4           [-1, 10, 15, 15]               0
            Conv2d-5           [-1, 16, 13, 13]           1,456
       BatchNorm2d-6           [-1, 16, 13, 13]              32
              ReLU-7           [-1, 16, 13, 13]               0
         MaxPool2d-8             [-1, 16, 6, 6]               0
            Conv2d-9             [-1, 32, 4, 4]           4,640
      BatchNorm2d-10             [-1, 32, 4, 4]              64
             ReLU-11             [-1, 32, 4, 4]               0
           Linear-12                   [-1, 64]          32,832
      BatchNorm1d-13                   [-1, 64]             128
             ReLU-14                   [-1, 64]               0
           Linear-15                   [-1, 16]           1,040
      BatchNorm1d-16                   [-1, 16]              32
             ReLU-17                   [-1, 16]               0
           Linear-18                   [-1, 10]             170
================================================================
Total params: 40,694
Trainable params: 40,694
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.30
Params size (MB): 0.16
Estimated Total Size (MB): 0.47
----------------------------------------------------------------
```

4. Training our model

In [16]:

```
def train(my_model, Epochs=20, Batch=2000, Data_workers=0, LR=0.1):

    # 1. Load our datasets
    train_data = torchvision.datasets.MNIST(root='./data/',train=True,download=True,tr
    test_data = torchvision.datasets.MNIST(root='./data/',train=False,download=True,tr
    train_loader = DataLoader(train_data, batch_size=Batch, shuffle=True, num_workers=
```

```
    test_loader  = DataLoader(test_data,  batch_size=Batch, shuffle=True, num_workers=

    print(train_data.classes)
    print(train_data.data.shape)
    print(test_data.data.shape)
    torch.cuda.is_available()
    Model = my_model().to(device)
    Parameters = sum(param.numel() for param in Model.parameters())
    print('Number of total parameters: ', Parameters)

    # 2. loss & optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(Model.parameters(), lr=LR, momentum=0.9)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size = 5, gamma = 0.8)

    # 3. Training
    train_loss_list = []
    test_loss_list  = []
    accuracy_list  = []
    lr_list = []
    w = []
    w_prime = []
    w_loss = []
    grad_list = []

    for epoch in range(Epochs):
        Model.train()
        train_loss = 0.0
        for i, data in enumerate(train_loader):
            images, labels = data
            images = (images.view(-1, 28*28)).to(device)
            labels = labels.to(device)
            outputs = Model(images)
            loss = criterion(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

    # 4. Evaluation
        Model.eval()
        with torch.no_grad():
            test_loss = 0
            correct = 0
            total = 0
            for data in test_loader:
                images, labels = data
                images = (images.view(-1, 28*28)).to(device)
                labels = labels.to(device)
                outputs = Model(images)
                loss = criterion(outputs, labels)
                test_loss += loss.item()
                _, pred = torch.max(outputs.data, 1)
                correct += (pred == labels).cpu().sum()
            total = len(test_loader.dataset)
            accuracy = 100.0*correct/total

    # 5. Save the losses values
        lr_list.append(optimizer.state_dict()['param_groups'][0]['lr'])
        train_loss_list.append(train_loss)
```

```
        test_loss_list.append(test_loss)
        accuracy_list.append(accuracy)
        print('{}/{} Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)'.format
                epoch, Epochs,test_loss, correct, total, accuracy))

    # 6. Weight collection
        if epoch % 1 == 0:
            # Layer weights
            weights_layer = np.zeros(0)
            for name,parameters in Model.named_parameters():
                if name == 'layer2.0.weight':
                    weight_i = (parameters.detach().cpu().numpy().reshape(-1))
                    weights_layer = np.concatenate((weights_layer, weight_i))
                    break
            print(weights_layer.shape)
            w_prime.append(weights_layer)

    # 7. Total number of weights
            weights = np.zeros(0)
            for name,parameters in Model.named_parameters():
                if name[-6:] == 'weight':
                    weight_i = (parameters.detach().cpu().numpy().reshape(-1))
                    weights = np.concatenate((weights, weight_i))
            print(weights.shape)
            w.append(weights)
            w_loss.append(train_loss)

    # 8. Gradient collection
        grad_all = 0.0
        for p in Model.parameters():
            grad = 0.0
            if p.grad is not None:
                grad = (p.grad.cpu().data.numpy()**2).sum()
            grad_all += grad
        grad_list.append(grad_all**0.5)

    return [train_loss_list,
            test_loss_list,
            accuracy_list,
            lr_list,
            w,
            w_prime,
            w_loss,
            grad_list]
```

5. Weight Collection

In [17]:

```
events = 8
W = []
W_loss = []
W_1 = []
G = []

for i in range(events):
    print('Event: '+str(i+1))
    [_,_,_,_,w,w_1,w_loss,grad_list] = train(DNN_MNIST)
    W.append(w)
    W_1.append(w_1)
```

```
    W_loss.append(w_loss)
    G.append(grad_list)
```

```
Event: 1
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
Number of total parameters:  25818
0/20 Test set: Average loss: 3.4754, Accuracy: 7707/10000 (77.07%)
(512,)
(25760,)
1/20 Test set: Average loss: 1.7935, Accuracy: 8948/10000 (89.48%)
(512,)
(25760,)
2/20 Test set: Average loss: 1.5520, Accuracy: 9124/10000 (91.24%)
(512,)
(25760,)
3/20 Test set: Average loss: 1.3186, Accuracy: 9225/10000 (92.25%)
(512,)
(25760,)
4/20 Test set: Average loss: 1.1891, Accuracy: 9304/10000 (93.04%)
(512,)
(25760,)
5/20 Test set: Average loss: 1.0555, Accuracy: 9369/10000 (93.69%)
(512,)
(25760,)
6/20 Test set: Average loss: 0.9739, Accuracy: 9425/10000 (94.25%)
(512,)
(25760,)
7/20 Test set: Average loss: 0.8982, Accuracy: 9463/10000 (94.63%)
(512,)
(25760,)
8/20 Test set: Average loss: 0.8590, Accuracy: 9476/10000 (94.76%)
(512,)
(25760,)
9/20 Test set: Average loss: 0.7953, Accuracy: 9517/10000 (95.17%)
(512,)
(25760,)
10/20 Test set: Average loss: 0.7407, Accuracy: 9554/10000 (95.54%)
(512,)
(25760,)
11/20 Test set: Average loss: 0.7180, Accuracy: 9576/10000 (95.76%)
(512,)
(25760,)
12/20 Test set: Average loss: 0.6875, Accuracy: 9583/10000 (95.83%)
(512,)
(25760,)
13/20 Test set: Average loss: 0.6480, Accuracy: 9619/10000 (96.19%)
(512,)
(25760,)
14/20 Test set: Average loss: 0.6492, Accuracy: 9614/10000 (96.14%)
(512,)
(25760,)
15/20 Test set: Average loss: 0.6452, Accuracy: 9608/10000 (96.08%)
(512,)
(25760,)
16/20 Test set: Average loss: 0.6020, Accuracy: 9633/10000 (96.33%)
(512,)
(25760,)
17/20 Test set: Average loss: 0.6094, Accuracy: 9629/10000 (96.29%)
(512,)
(25760,)
```

```
18/20 Test set: Average loss: 0.6026, Accuracy: 9638/10000 (96.38%)
(512,)
(25760,)
19/20 Test set: Average loss: 0.5974, Accuracy: 9656/10000 (96.56%)
(512,)
(25760,)
Event: 2
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
Number of total parameters:  25818
0/20 Test set: Average loss: 3.8449, Accuracy: 7352/10000 (73.52%)
(512,)
(25760,)
1/20 Test set: Average loss: 1.9356, Accuracy: 8915/10000 (89.15%)
(512,)
(25760,)
2/20 Test set: Average loss: 1.4522, Accuracy: 9195/10000 (91.95%)
(512,)
(25760,)
3/20 Test set: Average loss: 1.2284, Accuracy: 9279/10000 (92.79%)
(512,)
(25760,)
4/20 Test set: Average loss: 1.0983, Accuracy: 9366/10000 (93.66%)
(512,)
(25760,)
5/20 Test set: Average loss: 0.9982, Accuracy: 9406/10000 (94.06%)
(512,)
(25760,)
6/20 Test set: Average loss: 0.9391, Accuracy: 9437/10000 (94.37%)
(512,)
(25760,)
7/20 Test set: Average loss: 0.8430, Accuracy: 9507/10000 (95.07%)
(512,)
(25760,)
8/20 Test set: Average loss: 0.8338, Accuracy: 9495/10000 (94.95%)
(512,)
(25760,)
9/20 Test set: Average loss: 0.7845, Accuracy: 9534/10000 (95.34%)
(512,)
(25760,)
10/20 Test set: Average loss: 0.7618, Accuracy: 9555/10000 (95.55%)
(512,)
(25760,)
11/20 Test set: Average loss: 0.7341, Accuracy: 9551/10000 (95.51%)
(512,)
(25760,)
12/20 Test set: Average loss: 0.6963, Accuracy: 9580/10000 (95.80%)
(512,)
(25760,)
13/20 Test set: Average loss: 0.6911, Accuracy: 9584/10000 (95.84%)
(512,)
(25760,)
14/20 Test set: Average loss: 0.6805, Accuracy: 9593/10000 (95.93%)
(512,)
(25760,)
15/20 Test set: Average loss: 0.6495, Accuracy: 9621/10000 (96.21%)
(512,)
(25760,)
```

```
16/20 Test set: Average loss: 0.6665, Accuracy: 9612/10000 (96.12%)
(512,)
(25760,)
17/20 Test set: Average loss: 0.6590, Accuracy: 9603/10000 (96.03%)
(512,)
(25760,)
18/20 Test set: Average loss: 0.6231, Accuracy: 9630/10000 (96.30%)
(512,)
(25760,)
19/20 Test set: Average loss: 0.6275, Accuracy: 9631/10000 (96.31%)
(512,)
(25760,)
Event: 3
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
Number of total parameters:  25818
0/20 Test set: Average loss: 3.9410, Accuracy: 7459/10000 (74.59%)
(512,)
(25760,)
1/20 Test set: Average loss: 1.9008, Accuracy: 8887/10000 (88.87%)
(512,)
(25760,)
2/20 Test set: Average loss: 1.5243, Accuracy: 9102/10000 (91.02%)
(512,)
(25760,)
3/20 Test set: Average loss: 1.3436, Accuracy: 9178/10000 (91.78%)
(512,)
(25760,)
4/20 Test set: Average loss: 1.1986, Accuracy: 9291/10000 (92.91%)
(512,)
(25760,)
5/20 Test set: Average loss: 1.1135, Accuracy: 9341/10000 (93.41%)
(512,)
(25760,)
6/20 Test set: Average loss: 1.0339, Accuracy: 9403/10000 (94.03%)
(512,)
(25760,)
7/20 Test set: Average loss: 0.9906, Accuracy: 9399/10000 (93.99%)
(512,)
(25760,)
8/20 Test set: Average loss: 0.9000, Accuracy: 9472/10000 (94.72%)
(512,)
(25760,)
9/20 Test set: Average loss: 0.8753, Accuracy: 9494/10000 (94.94%)
(512,)
(25760,)
10/20 Test set: Average loss: 0.8143, Accuracy: 9511/10000 (95.11%)
(512,)
(25760,)
11/20 Test set: Average loss: 0.7854, Accuracy: 9523/10000 (95.23%)
(512,)
(25760,)
12/20 Test set: Average loss: 0.7461, Accuracy: 9556/10000 (95.56%)
(512,)
(25760,)
13/20 Test set: Average loss: 0.7561, Accuracy: 9547/10000 (95.47%)
(512,)
(25760,)
```

```
14/20 Test set: Average loss: 0.7169, Accuracy: 9576/10000 (95.76%)
(512,)
(25760,)
15/20 Test set: Average loss: 0.6949, Accuracy: 9590/10000 (95.90%)
(512,)
(25760,)
16/20 Test set: Average loss: 0.6667, Accuracy: 9606/10000 (96.06%)
(512,)
(25760,)
17/20 Test set: Average loss: 0.6958, Accuracy: 9595/10000 (95.95%)
(512,)
(25760,)
18/20 Test set: Average loss: 0.7299, Accuracy: 9581/10000 (95.81%)
(512,)
(25760,)
19/20 Test set: Average loss: 0.6575, Accuracy: 9625/10000 (96.25%)
(512,)
(25760,)
Event: 4
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
Number of total parameters:  25818
0/20 Test set: Average loss: 3.9467, Accuracy: 7451/10000 (74.51%)
(512,)
(25760,)
1/20 Test set: Average loss: 1.8207, Accuracy: 8934/10000 (89.34%)
(512,)
(25760,)
2/20 Test set: Average loss: 1.4887, Accuracy: 9162/10000 (91.62%)
(512,)
(25760,)
3/20 Test set: Average loss: 1.2973, Accuracy: 9250/10000 (92.50%)
(512,)
(25760,)
4/20 Test set: Average loss: 1.1831, Accuracy: 9307/10000 (93.07%)
(512,)
(25760,)
5/20 Test set: Average loss: 1.0857, Accuracy: 9382/10000 (93.82%)
(512,)
(25760,)
6/20 Test set: Average loss: 0.9740, Accuracy: 9403/10000 (94.03%)
(512,)
(25760,)
7/20 Test set: Average loss: 0.8861, Accuracy: 9493/10000 (94.93%)
(512,)
(25760,)
8/20 Test set: Average loss: 0.8419, Accuracy: 9522/10000 (95.22%)
(512,)
(25760,)
9/20 Test set: Average loss: 0.8109, Accuracy: 9495/10000 (94.95%)
(512,)
(25760,)
10/20 Test set: Average loss: 0.7435, Accuracy: 9542/10000 (95.42%)
(512,)
(25760,)
11/20 Test set: Average loss: 0.7414, Accuracy: 9560/10000 (95.60%)
(512,)
(25760,)
```

```
12/20 Test set: Average loss: 0.6810, Accuracy: 9592/10000 (95.92%)
(512,)
(25760,)
13/20 Test set: Average loss: 0.6603, Accuracy: 9617/10000 (96.17%)
(512,)
(25760,)
14/20 Test set: Average loss: 0.6780, Accuracy: 9607/10000 (96.07%)
(512,)
(25760,)
15/20 Test set: Average loss: 0.6544, Accuracy: 9619/10000 (96.19%)
(512,)
(25760,)
16/20 Test set: Average loss: 0.6243, Accuracy: 9638/10000 (96.38%)
(512,)
(25760,)
17/20 Test set: Average loss: 0.6537, Accuracy: 9605/10000 (96.05%)
(512,)
(25760,)
18/20 Test set: Average loss: 0.6141, Accuracy: 9643/10000 (96.43%)
(512,)
(25760,)
19/20 Test set: Average loss: 0.6058, Accuracy: 9640/10000 (96.40%)
(512,)
(25760,)
Event: 5
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
Number of total parameters:  25818
0/20 Test set: Average loss: 3.5873, Accuracy: 7680/10000 (76.80%)
(512,)
(25760,)
1/20 Test set: Average loss: 1.7761, Accuracy: 8923/10000 (89.23%)
(512,)
(25760,)
2/20 Test set: Average loss: 1.3950, Accuracy: 9179/10000 (91.79%)
(512,)
(25760,)
3/20 Test set: Average loss: 1.1876, Accuracy: 9313/10000 (93.13%)
(512,)
(25760,)
4/20 Test set: Average loss: 1.0792, Accuracy: 9376/10000 (93.76%)
(512,)
(25760,)
5/20 Test set: Average loss: 0.9546, Accuracy: 9443/10000 (94.43%)
(512,)
(25760,)
6/20 Test set: Average loss: 0.8834, Accuracy: 9469/10000 (94.69%)
(512,)
(25760,)
7/20 Test set: Average loss: 0.8068, Accuracy: 9508/10000 (95.08%)
(512,)
(25760,)
8/20 Test set: Average loss: 0.7855, Accuracy: 9538/10000 (95.38%)
(512,)
(25760,)
9/20 Test set: Average loss: 0.7563, Accuracy: 9563/10000 (95.63%)
(512,)
(25760,)
```

```
10/20 Test set: Average loss: 0.6791, Accuracy: 9602/10000 (96.02%)
(512,)
(25760,)
11/20 Test set: Average loss: 0.6949, Accuracy: 9596/10000 (95.96%)
(512,)
(25760,)
12/20 Test set: Average loss: 0.6694, Accuracy: 9620/10000 (96.20%)
(512,)
(25760,)
13/20 Test set: Average loss: 0.6576, Accuracy: 9601/10000 (96.01%)
(512,)
(25760,)
14/20 Test set: Average loss: 0.6070, Accuracy: 9647/10000 (96.47%)
(512,)
(25760,)
15/20 Test set: Average loss: 0.6016, Accuracy: 9658/10000 (96.58%)
(512,)
(25760,)
16/20 Test set: Average loss: 0.5958, Accuracy: 9665/10000 (96.65%)
(512,)
(25760,)
17/20 Test set: Average loss: 0.6076, Accuracy: 9649/10000 (96.49%)
(512,)
(25760,)
18/20 Test set: Average loss: 0.5755, Accuracy: 9664/10000 (96.64%)
(512,)
(25760,)
19/20 Test set: Average loss: 0.5725, Accuracy: 9655/10000 (96.55%)
(512,)
(25760,)
Event: 6
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
Number of total parameters:  25818
0/20 Test set: Average loss: 3.5877, Accuracy: 7792/10000 (77.92%)
(512,)
(25760,)
1/20 Test set: Average loss: 1.9113, Accuracy: 8922/10000 (89.22%)
(512,)
(25760,)
2/20 Test set: Average loss: 1.4979, Accuracy: 9105/10000 (91.05%)
(512,)
(25760,)
3/20 Test set: Average loss: 1.3241, Accuracy: 9225/10000 (92.25%)
(512,)
(25760,)
4/20 Test set: Average loss: 1.2094, Accuracy: 9288/10000 (92.88%)
(512,)
(25760,)
5/20 Test set: Average loss: 1.1149, Accuracy: 9339/10000 (93.39%)
(512,)
(25760,)
6/20 Test set: Average loss: 1.0400, Accuracy: 9374/10000 (93.74%)
(512,)
(25760,)
7/20 Test set: Average loss: 0.9532, Accuracy: 9432/10000 (94.32%)
(512,)
(25760,)
```

```
8/20 Test set: Average loss: 0.9045, Accuracy: 9457/10000 (94.57%)
(512,)
(25760,)
9/20 Test set: Average loss: 0.8418, Accuracy: 9491/10000 (94.91%)
(512,)
(25760,)
10/20 Test set: Average loss: 0.8050, Accuracy: 9497/10000 (94.97%)
(512,)
(25760,)
11/20 Test set: Average loss: 0.7436, Accuracy: 9553/10000 (95.53%)
(512,)
(25760,)
12/20 Test set: Average loss: 0.7021, Accuracy: 9543/10000 (95.43%)
(512,)
(25760,)
13/20 Test set: Average loss: 0.7052, Accuracy: 9567/10000 (95.67%)
(512,)
(25760,)
14/20 Test set: Average loss: 0.6546, Accuracy: 9613/10000 (96.13%)
(512,)
(25760,)
15/20 Test set: Average loss: 0.6378, Accuracy: 9605/10000 (96.05%)
(512,)
(25760,)
16/20 Test set: Average loss: 0.6327, Accuracy: 9625/10000 (96.25%)
(512,)
(25760,)
17/20 Test set: Average loss: 0.6385, Accuracy: 9604/10000 (96.04%)
(512,)
(25760,)
18/20 Test set: Average loss: 0.6133, Accuracy: 9629/10000 (96.29%)
(512,)
(25760,)
19/20 Test set: Average loss: 0.6064, Accuracy: 9634/10000 (96.34%)
(512,)
(25760,)
Event: 7
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
Number of total parameters:  25818
0/20 Test set: Average loss: 5.5359, Accuracy: 6673/10000 (66.73%)
(512,)
(25760,)
1/20 Test set: Average loss: 2.0856, Accuracy: 8755/10000 (87.55%)
(512,)
(25760,)
2/20 Test set: Average loss: 1.5279, Accuracy: 9107/10000 (91.07%)
(512,)
(25760,)
3/20 Test set: Average loss: 1.2697, Accuracy: 9276/10000 (92.76%)
(512,)
(25760,)
4/20 Test set: Average loss: 1.1229, Accuracy: 9371/10000 (93.71%)
(512,)
(25760,)
5/20 Test set: Average loss: 1.0459, Accuracy: 9399/10000 (93.99%)
(512,)
(25760,)
```

```
6/20 Test set: Average loss: 0.9170, Accuracy: 9480/10000 (94.80%)
(512,)
(25760,)
7/20 Test set: Average loss: 0.8604, Accuracy: 9501/10000 (95.01%)
(512,)
(25760,)
8/20 Test set: Average loss: 0.8086, Accuracy: 9527/10000 (95.27%)
(512,)
(25760,)
9/20 Test set: Average loss: 0.7741, Accuracy: 9543/10000 (95.43%)
(512,)
(25760,)
10/20 Test set: Average loss: 0.7389, Accuracy: 9568/10000 (95.68%)
(512,)
(25760,)
11/20 Test set: Average loss: 0.7215, Accuracy: 9583/10000 (95.83%)
(512,)
(25760,)
12/20 Test set: Average loss: 0.7016, Accuracy: 9589/10000 (95.89%)
(512,)
(25760,)
13/20 Test set: Average loss: 0.6506, Accuracy: 9613/10000 (96.13%)
(512,)
(25760,)
14/20 Test set: Average loss: 0.6418, Accuracy: 9621/10000 (96.21%)
(512,)
(25760,)
15/20 Test set: Average loss: 0.6414, Accuracy: 9636/10000 (96.36%)
(512,)
(25760,)
16/20 Test set: Average loss: 0.6031, Accuracy: 9635/10000 (96.35%)
(512,)
(25760,)
17/20 Test set: Average loss: 0.6285, Accuracy: 9610/10000 (96.10%)
(512,)
(25760,)
18/20 Test set: Average loss: 0.5944, Accuracy: 9649/10000 (96.49%)
(512,)
(25760,)
19/20 Test set: Average loss: 0.5944, Accuracy: 9656/10000 (96.56%)
(512,)
(25760,)
Event: 8
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
Number of total parameters:  25818
0/20 Test set: Average loss: 3.6821, Accuracy: 7621/10000 (76.21%)
(512,)
(25760,)
1/20 Test set: Average loss: 1.9533, Accuracy: 8857/10000 (88.57%)
(512,)
(25760,)
2/20 Test set: Average loss: 1.6545, Accuracy: 9031/10000 (90.31%)
(512,)
(25760,)
3/20 Test set: Average loss: 1.3576, Accuracy: 9188/10000 (91.88%)
(512,)
(25760,)
```

```
4/20 Test set: Average loss: 1.1927, Accuracy: 9278/10000 (92.78%)
(512,)
(25760,)
5/20 Test set: Average loss: 1.0649, Accuracy: 9365/10000 (93.65%)
(512,)
(25760,)
6/20 Test set: Average loss: 0.9786, Accuracy: 9404/10000 (94.04%)
(512,)
(25760,)
7/20 Test set: Average loss: 0.9166, Accuracy: 9453/10000 (94.53%)
(512,)
(25760,)
8/20 Test set: Average loss: 0.8607, Accuracy: 9482/10000 (94.82%)
(512,)
(25760,)
9/20 Test set: Average loss: 0.8236, Accuracy: 9502/10000 (95.02%)
(512,)
(25760,)
10/20 Test set: Average loss: 0.7431, Accuracy: 9548/10000 (95.48%)
(512,)
(25760,)
11/20 Test set: Average loss: 0.7284, Accuracy: 9552/10000 (95.52%)
(512,)
(25760,)
12/20 Test set: Average loss: 0.7194, Accuracy: 9569/10000 (95.69%)
(512,)
(25760,)
13/20 Test set: Average loss: 0.6708, Accuracy: 9590/10000 (95.90%)
(512,)
(25760,)
14/20 Test set: Average loss: 0.6454, Accuracy: 9604/10000 (96.04%)
(512,)
(25760,)
15/20 Test set: Average loss: 0.6672, Accuracy: 9586/10000 (95.86%)
(512,)
(25760,)
16/20 Test set: Average loss: 0.6434, Accuracy: 9609/10000 (96.09%)
(512,)
(25760,)
17/20 Test set: Average loss: 0.6442, Accuracy: 9606/10000 (96.06%)
(512,)
(25760,)
18/20 Test set: Average loss: 0.6194, Accuracy: 9614/10000 (96.14%)
(512,)
(25760,)
19/20 Test set: Average loss: 0.6109, Accuracy: 9622/10000 (96.22%)
(512,)
(25760,)
```

6. Dimention Reduction using PCA

In [22]:

```python
def W_Prime_PCA(W_i):
    w = np.array(W_i)
    pca = PCA(n_components=2)
    pca.fit(w)
    w_new = pca.transform(w)
    return w_new

W1 = []
```

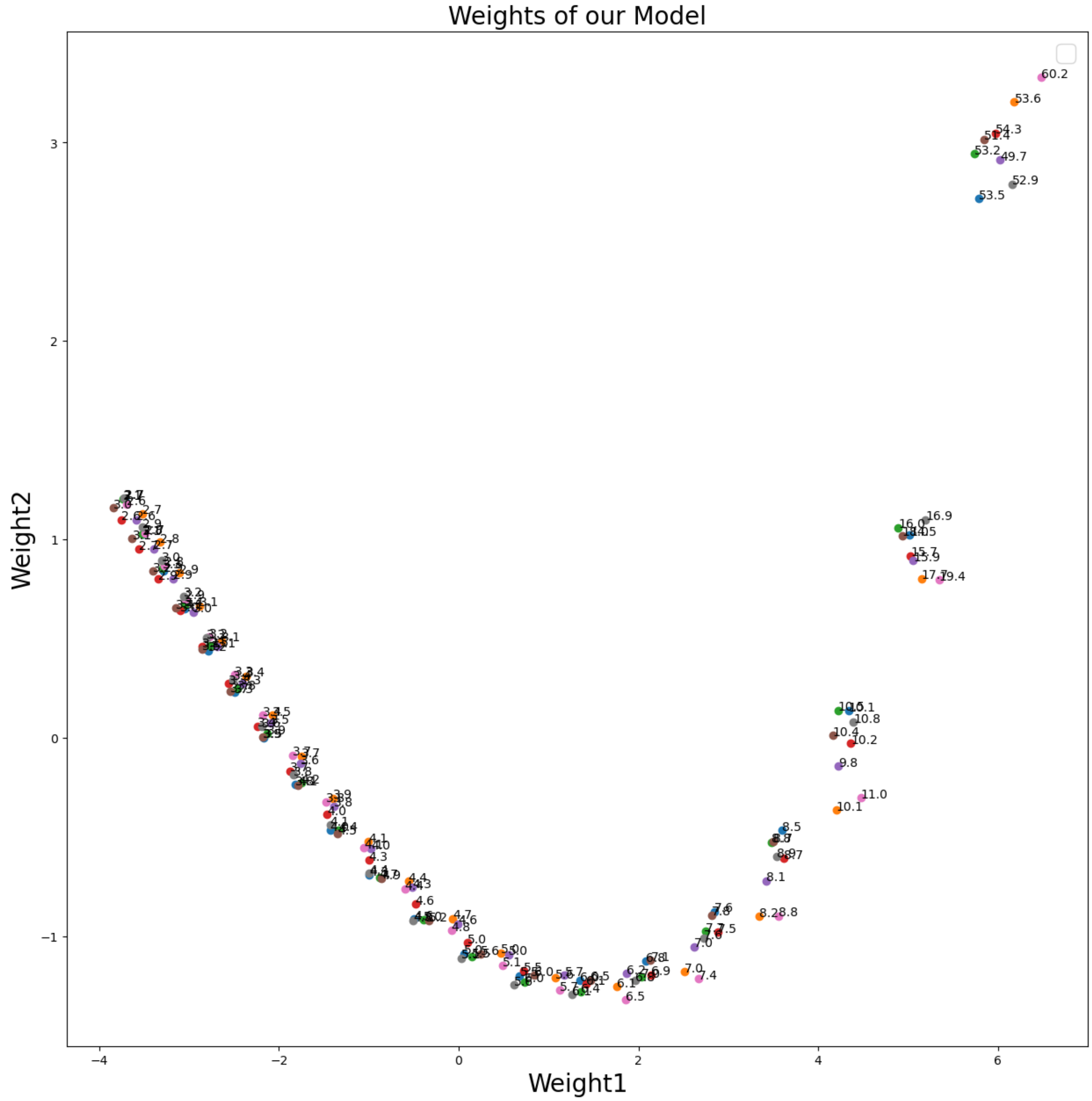
```
for i in range(events):
    W0 = W_Prime_PCA(W[i])
    W1.append(W0)
W1 = np.array(W1)

W2 = []
for i in range(events):
    W00 = W_Prime_PCA(W_1[i])
    W2.append(W00)
W2 = np.array(W2)
```

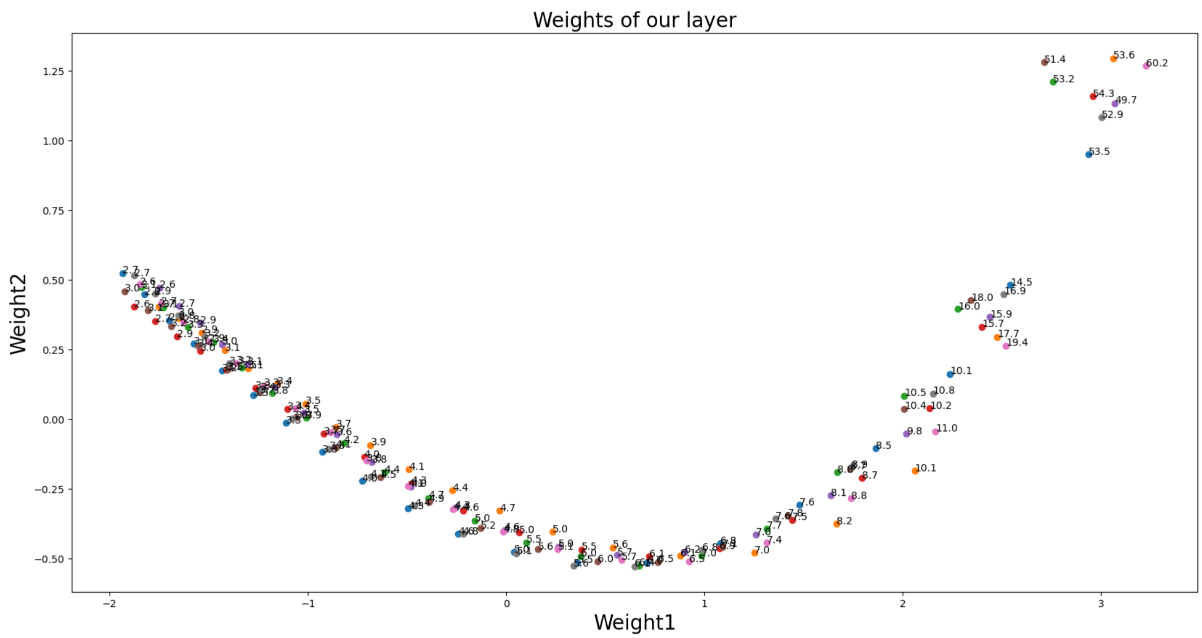
7. Ploting and Visualization Optimization Process

In [25]:

```
plt.figure(figsize=(15,15))
plt.xlabel('Weight1',fontsize=20)
plt.ylabel('Weight2',fontsize=20)
plt.title('Weights of our Model',fontsize=20)
plt.legend(fontsize=20)

for i in range(events):
    W_i = W1[i]
    plt.scatter(W_i[:,0], W_i[:,1])

    for j in range(len(W_i)):
        plt.annotate(round(W_loss[i][j],1), (W_i[j,0],W_i[j,1]))
plt.show()
plt.figure(figsize=(20,10))
plt.xlabel('Weight1',fontsize=20)
plt.ylabel('Weight2',fontsize=20)
plt.title('Weights of our layer',fontsize=20)

for i in range(events):
    W_i = W2[i]
    plt.scatter(W_i[:,0], W_i[:,1])

    for j in range(len(W_i)):
        plt.annotate(round(W_loss[i][j],1), (W_i[j,0],W_i[j,1]))
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start
with an underscore are ignored when legend() is called with no argument.
```

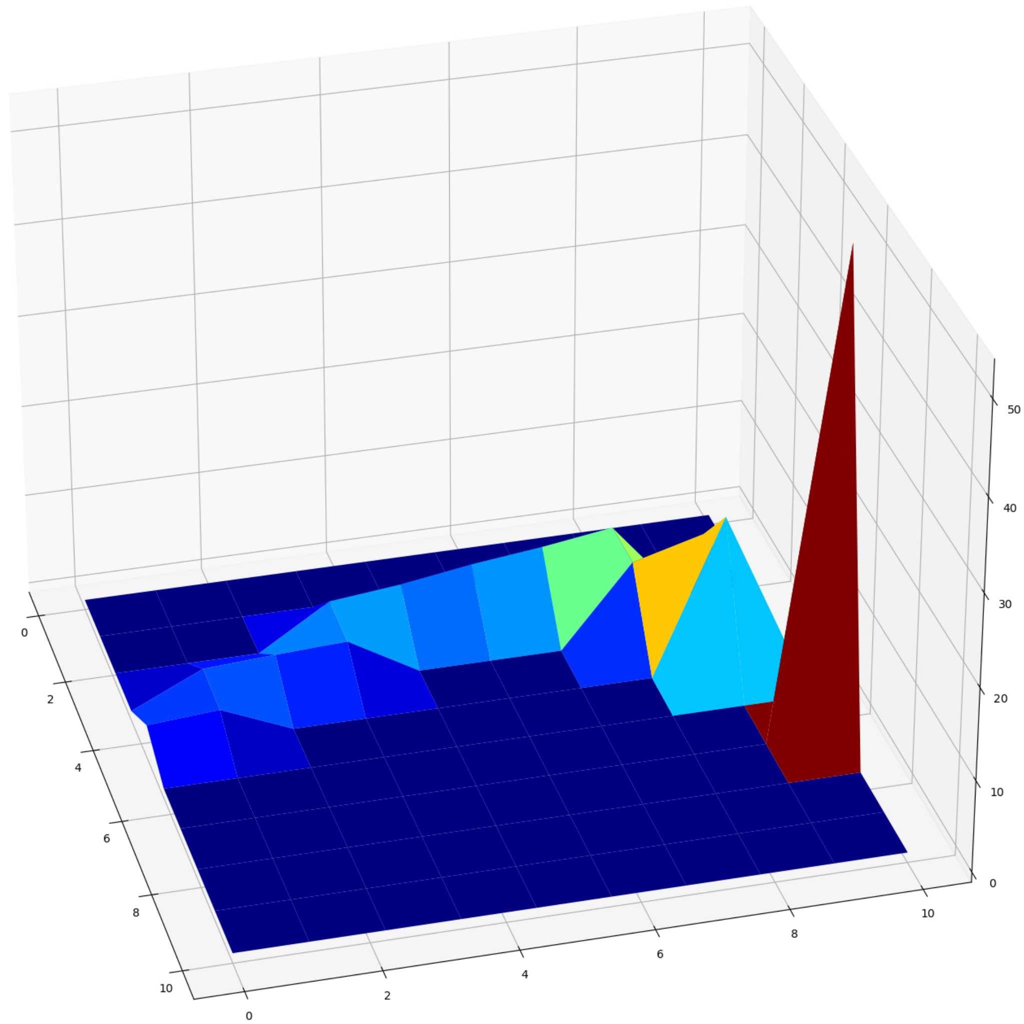




8. Visualization Error Surface

In [30]:
```
fig = plt.figure(figsize=(15,15))
ax = Axes3D(fig)

W_all = W1.reshape((events*len(W2[0]),2))
W_total = np.array(W_loss)
W_total = W_total.reshape(-1)

x = np.linspace(0,10,10)
y = np.linspace(0,10,10)
X, Y = np.meshgrid(x, y)
z = X-X

error = (W_all-np.min(W_all))/(np.max(W_all)-np.min(W_all))
error[error==1] = 0.99

for i in range(len(error[:,0])):
    zx = ((error[i,0])*10).astype(np.int)
    zy = ((error[i,1])*10).astype(np.int)
    z[zx,zy] = W_total[i]

ax.plot_surface(X, Y, z, cmap='jet')
ax.view_init(elev=30, azim=-15)
plt.show()
```

```
C:\Users\shaerib\AppData\Local\Temp\ipykernel_13920\2546280402.py:2: MatplotlibDeprec
ationWarning: Axes3D(fig) adding itself to the figure is deprecated since 3.4. Pass t
he keyword argument auto_add_to_figure=False and use fig.add_axes(ax) to suppress thi
s warning. The default value of auto_add_to_figure will change to False in mpl3.5 and
True values will no longer work in 3.6.  This is consistent with other Axes classes.
  ax = Axes3D(fig)
C:\Users\shaerib\AppData\Local\Temp\ipykernel_13920\2546280402.py:17: DeprecationWarn
ing: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, u
se `int` by itself. Doing this will not modify any behavior and is safe. When replaci
ng `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precisi
on. If you wish to review your current use, check the release note link for additiona
l information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/re
lease/1.20.0-notes.html#deprecations
  zx = ((error[i,0])*10).astype(np.int)
C:\Users\shaerib\AppData\Local\Temp\ipykernel_13920\2546280402.py:18: DeprecationWarn
ing: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, u
se `int` by itself. Doing this will not modify any behavior and is safe. When replaci
ng `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precisi
on. If you wish to review your current use, check the release note link for additiona
l information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/re
lease/1.20.0-notes.html#deprecations
  zy = ((error[i,1])*10).astype(np.int)
```



9. Observe Gradient Norm During Training

In [31]:
```
class DNN_MNIST_3L(nn.Module):
    def __init__(self):
        super(DNN_MNIST_3L, self).__init__()
        self.layer1 = nn.Sequential(nn.Linear(1, 32),nn.ReLU(True))
        self.layer2 = nn.Sequential(nn.Linear(32, 16),nn.ReLU(True))
        self.layer3 = nn.Sequential(nn.Linear(16, 1))
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        return x

device = torch.device("cuda")
Model = DNN_MNIST_3L().to(device)
summary(Model, (1,1))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 32]              64
              ReLU-2                [-1, 1, 32]               0
            Linear-3                [-1, 1, 16]             528
              ReLU-4                [-1, 1, 16]               0
            Linear-5                 [-1, 1, 1]              17
================================================================
Total params: 609
Trainable params: 609
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

In [36]:

```python
# 1. Initialization
x = torch.linspace(0,1,1000).unsqueeze(1)
y = torch.sin(5*np.pi*x)/(5*np.pi*x)
y[0] = y[1]
function1 = y


# 2. Define train function
def train(function,
          model_name,
          Epochs = 20000,
          Batch  = 1000,
          Data_workers = 0,
          LR = 0.0005):

# 3. Initialization model
    torch.cuda.is_available()
    Model = model_name().to(device)
    x = torch.linspace(0,1,1000).unsqueeze(1)
    x = x.to(device)
    y = function.to(device)

# 4. loss & optimizer
    criterion = nn.MSELoss()
    optimizer = optim.Adam(Model.parameters(), lr=LR)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size = 100, gamma = 0.8)

# 5. Training
    train_loss_list = []
    lr_list = []
    grad_list = []

    for epoch in range(Epochs):
        Model.train()
        train_loss = 0.0
        y_pred = Model(x)
        loss = criterion(y_pred, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss = loss.item()
```

```
        train_loss_list.append(train_loss)

        if epoch % (Epochs//10) == 0:
            print('{}/{}, loss: {}'.format(epoch,Epochs,train_loss))
        lr_list.append(optimizer.state_dict()['param_groups'][0]['lr'])


# 6. Grad collect
        grad_all = 0.0
        for p in Model.parameters():
            grad = 0.0
            if p.grad is not None:
                grad = (p.grad.cpu().data.numpy()**2).sum()
            grad_all += grad
        grad_list.append(grad_all**0.5)

    return [Model,train_loss_list,lr_list,grad_list]
```

In [37]:

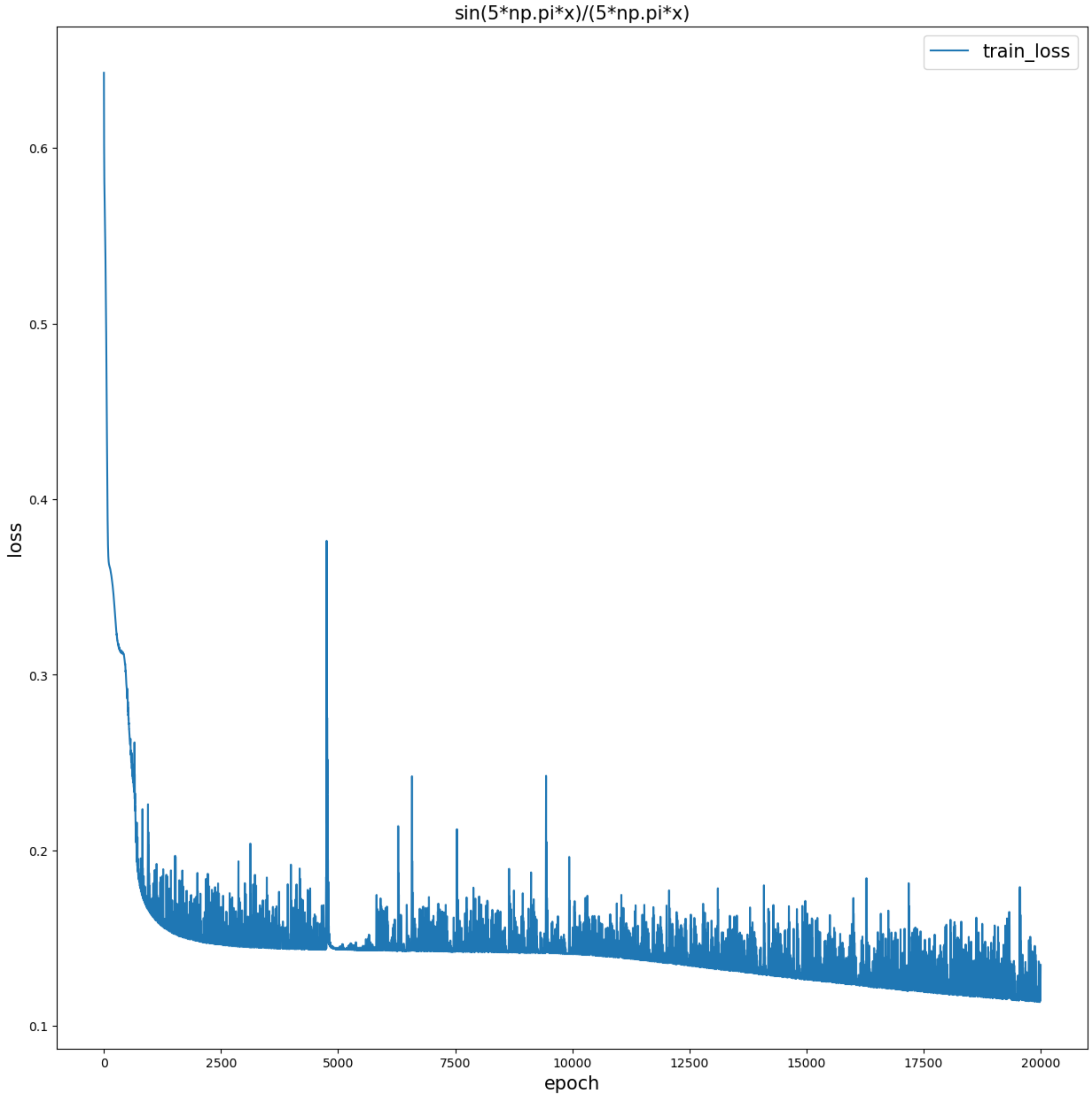
```
[Model,trainloss,lr,grad_list] = train(function1, DNN_MNIST_3L, Epochs=20000, Batch=1(
```

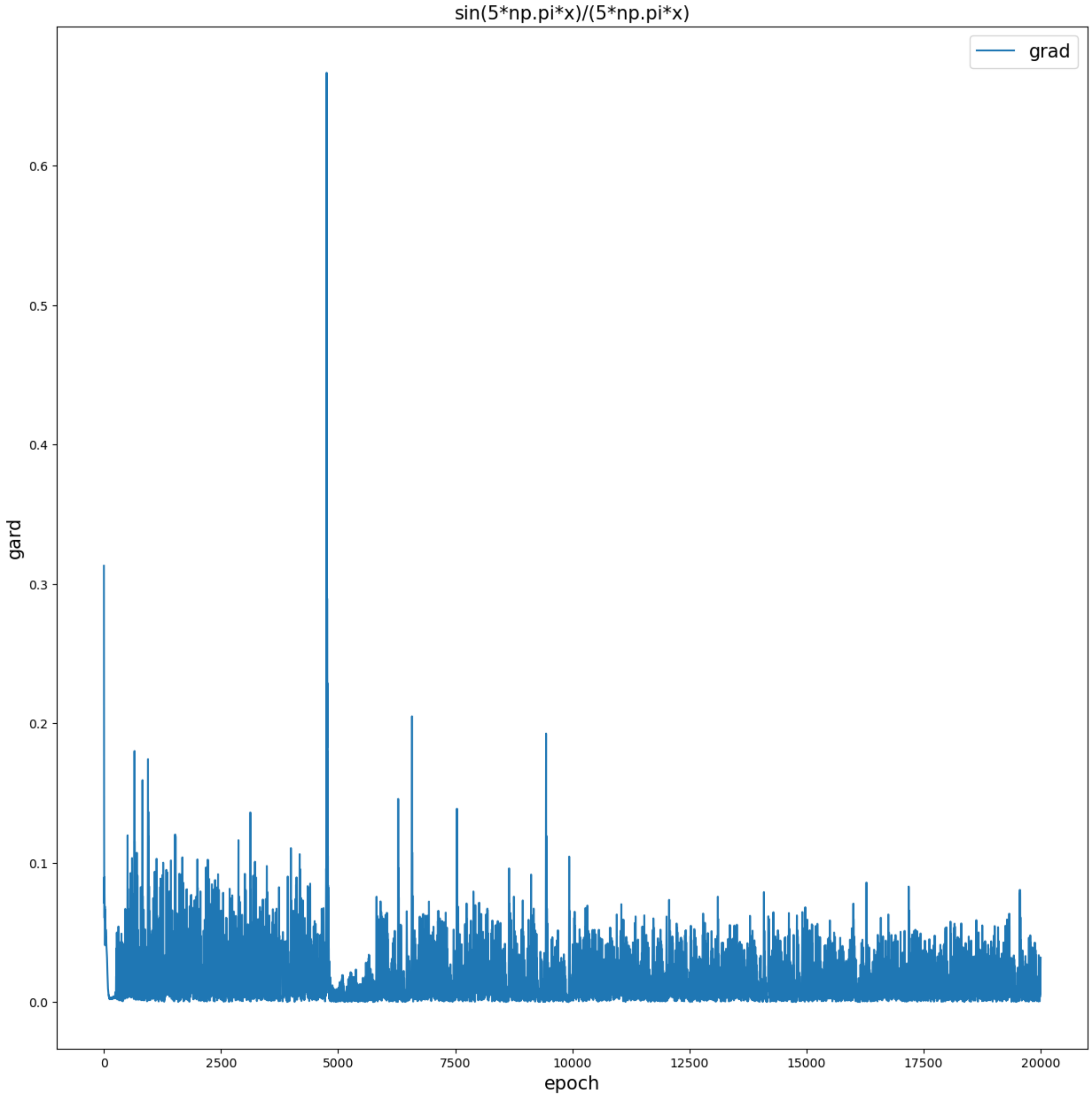
```
0/20000, loss: 0.10970278084278107
2000/20000, loss: 7.86061937105842e-05
4000/20000, loss: 8.000803063623607e-05
6000/20000, loss: 6.482381286332384e-05
8000/20000, loss: 9.42010447033681e-05
10000/20000, loss: 5.641512689180672e-05
12000/20000, loss: 4.644765067496337e-05
14000/20000, loss: 3.641786679509096e-05
16000/20000, loss: 0.00015428826736751944
18000/20000, loss: 3.202572406735271e-05
```

10. Plotting the gradient norm and the loss

In [38]:

```
plt.figure(figsize=(15,15))
plt.plot(np.array(trainloss)**0.2, label='train_loss')
plt.xlabel('epoch',fontsize=15)
plt.ylabel('loss',fontsize=15)
plt.title('sin(5*np.pi*x)/(5*np.pi*x)',fontsize=15)
plt.legend(fontsize=15)
plt.show()

plt.figure(figsize=(15,15))
plt.plot(np.array(grad_list)**1, label='grad')
plt.xlabel('epoch',fontsize=15)
plt.ylabel('gard',fontsize=15)
plt.title('sin(5*np.pi*x)/(5*np.pi*x)',fontsize=15)
plt.legend(fontsize=15)
plt.show()
```


sin(5*np.pi*x)/(5*np.pi*x)
train_loss
loss
epoch
0.1
0.2
0.3
0.4
0.5
0.6
0
2500
5000
7500
10000
12500
15000
17500
20000


sin(5*np.pi*x)/(5*np.pi*x)
grad
gard
epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
2500
5000
7500
10000
12500
15000
17500
20000

In [ ]: